# Towards the Automatic Evolutionary Prediction of the FOREX Market Behaviour

Karel Slaný
Department of Computer Systems
Faculty of Information Technology, Brno University of Technology
Brno, Czech Republic
Email: slany@fit.vutbr.cz

## Abstract

*In this paper a self-adapting architecture for FOREX market prediction, which is being developed, is described. The proposed system utilizes genetic programming (GP) for predictor representation. The goal of the system is the design and adaptation of simple predictors which can either be used by the system itself or be 'manually' used by a human trader.*

## Keywords

*FOREX; prediction; genetic programming; on-line evolution*

## 1. Introduction

The FOREX is the worlds biggest market. The trading moves around the globe 24 hours a day, 7 days a week. It is considered to trade the amount of $1.5 \cdot 10^{12}$USD every day. Formerly the market was accessible only to banks and big financial institutions. Nowadays everyone can open an account at a broker and go trading FOREX commodities. However with many big institutions and even more small traders, all of them using computers to somehow predict the future behaviour, the market is considered to be very efficient. The market by itself provides less information about its future behaviour. Synthetic prediction models based on various techniques such as statistical analysis [2], [4], [6], neural networks, fuzzy logic [1], evolutionary algorithms [3] etc. are constantly being developed [5].

Evolutionary algorithms (EAs) represent an interesting alternative to conventional design techniques. This is because EAs can generate solutions without the knowledge about the internal structure of the solved problem. The found solutions frequently embody complex structures which are difficult to understand by a human designer. In the case of FOREX data prediction this can be of advantage, because the market shows a very complicated behaviour. The EA can be used either to optimize parameters of a human-designed model or to design the predictor model from scratch.

Another way of classifying the design process is according to the training process. The predictor can be designed and verified on static data before being used. After the design process is finished the predictor is going to be used. After some time a demand for a new predictor will rise, because the currently used predictor is obsolete. Then the process of evolution needs to be restarted. In the second case the predictor is being trained on dynamically changing training data. The predictors, designed during the never-ending adaptation process, are constantly being optimized to meet the changes in the environment. The currently best evolved predictor can be verified and used for prediction. It can be automatically replaced by a fitter solution.

In the paper a self-adapting system architecture is being described. The system is based on a EA running in background to the changing environment.

## 2. System Structure

The evolutionary system, in it's structure, is designed to operate independently on the human-operator interference. It is capable of adapting the evolved candidate solutions to the changes in the market environment on the fly. The design follows the design of a more simple system [12] which is also working in a changing environment. The system can be divided into five separately operating subsystems (fig. 1). Currently it is used for prediction of FOREX market behaviour but it can be easily modified to deal with other tasks.

### 2.1. Preprocessor

The preprocessor is responsible for translating input data into an internal representation. The system uses a low-level input data representation - tick data. The data are obtained by sampling price changes. They are in no way modified and enter the system as they are sampled from the market.

The main idea of the preprocessor is not filtering-out the possible noise from the input data because it might contain useful information. The preprocessor, as it is designed, introduces additional information into the input data. Additional supportive values are computed, such as moving averages of different time-windows. The values can pick out interesting data such as peaks in specified time intervals. The preprocessor also computes, with a certain delay, the referential function which is used for fitness evaluation. The preprocessor is primary designed to reduce the computational load of the EA unit, simply by computing some built-in function results in advance – this can speed-up the evolution heavily.

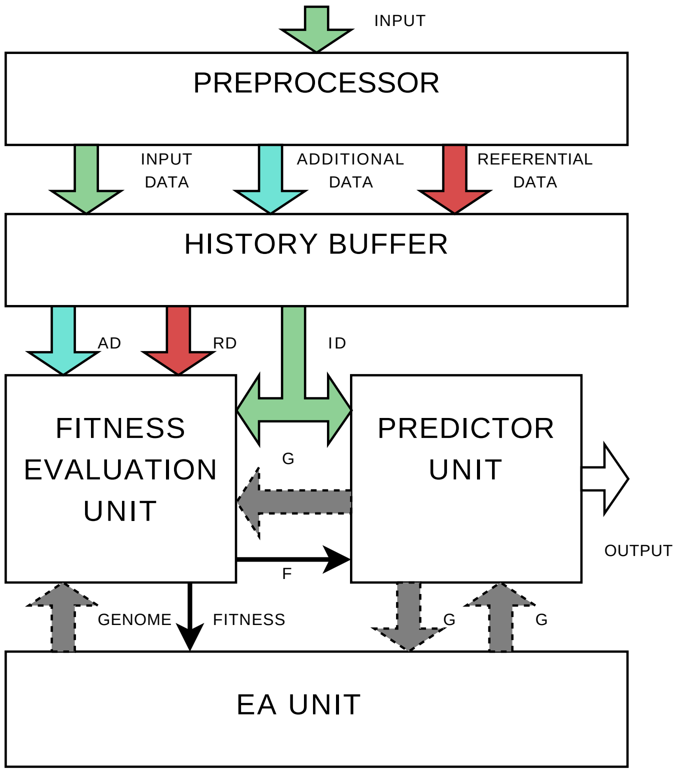

Fig. 1. Structure of the predictor system. The system is designed to run EAs in changing environments. The core of the system is the EA unit communicating with the predictor unit which containing the best evolved solutions and computing the system response.

To evaluate all the necessary data to support the evolution the preprocessor contains a circular buffer of sampled past inputs which are being used to compute all the necessary data.

All the data produced by the preprocessor are supplied with a time stamp. The time stamp serves for the easier assemblage of the data in other system units.

### 2.2. History Buffer

The preprocessed data are provided with a time stamp and then stored into a large buffer according to the supplied time stamps. Old data are replaced with new one. The data stored in the buffer are being used both as training data for the EA and also as input data for the prediction unit.

As mentioned before, the preprocessor unit can compute the referential function for training data but with a certain delay. The history buffer has to assemble the correct referential function values with the sampled tick data which are already stored in the buffer. This can be done easily according to the time stamps.

At the current stage of the system development the time stamps are only used to identify corresponding data in the buffer. But there exists a possibility for utilizing the information in the time stamps. This is one of the subjects of the experiments in progress.

### 2.3. EA Unit

The core of the system is a unit capable of running evolutionary algorithms which designs new predictors and/or adapts them to the continually changing environment. The system utilizes tree-based genetic programming (GP) chromosomes [9] for predictor representation. The tree-based GP representation can easily be changed to another form of GP such as Cartesian genetic programming (CGP) [11]. This is an advantage in cases where a fixed chromosome length is desired.

The unit holds a population of candidate solutions which are constantly being adapted to the changing environment. The fitness of the best solution is being compared with the fitness values of the predictors stored in the prediction unit. In cases when a candidate solution is better than a solution in the predictor unit a candidate solution is send/received to/from the predictor unit.

### 2.4. Fitness Evaluation Unit

The fitness evaluation is located separately from the EA unit because fitness values are also needed for the predictor evaluation in the prediction unit. The unit communicates with the history buffer, which contains the training data, and evaluates the fitness value of all the candidate solutions and predictors which are present in the system.

The fitness evaluation unit is the most resource demanding subsystem because of the nature of the training data. Each candidate solution and predictor is being evaluated on millions tick data samples.

### 2.5. Predictor Unit

This is the executive unit of the system. The system outputs are generated here. The unit is designed to utilize the evolved candidate solutions and compute the systems response. The unit communicates with the history buffer and the EA unit. It holds the best evolved predictors and uses them to compute the systems response. It is designed to use several predictors in parallel for better prediction accuracy. However it can be run also in a 'single mode' where only one predictor is used.

## 3. Implementation Details

The market data can be seen as a time series of chronologically ordered observations of a specific event. One of the essential attributes is the mutual dependency of the observed values. The analysis of such series constitutes of finding a model showing how mutually dependent the observed values are.

The prediction of a time series is a process of finding the most accurate guess (at the time $t$) of the future value (at the time $t+l$). The guess is computed from the past sampled data at the time $t$. The time series can be described as a sequence of values $z_{t-n+1}, z_{t-n+2}, \ldots, z_{t-1}, z_t$, where $z_i \in R$. The notation $\hat{z}_t(l)$ represents the forecast of $z_{t+l}$ made in time $t$.

## 3.1. Data Structure

The described system is not designed to predict future market prices directly. At the current stage of development the system is used for indication of turning points - to identify major trend changes. These can be identified easily in the past data when enough data is available. Nevertheless the identification whether the market is currently approaching a turning point is a more difficult task.

The system operates at a very low level. The input are tick data i.e. unfiltered data sampled directly from the market. The data are processed by the preprocessing unit where the turning points in the past data are identified, marked and added as referential values to the training data.

The process of the computation of the referential data starts with dividing the training data into sub-sequences $W$ of equal length, usually a week. Each sequence $W$ is divided into $n$ disjunct sub-sequences of equal size $W = W_0 \cup W_1 \cup \ldots \cup W_{n-1}$. The function

$$IdxW(t_s) = i, z_{t_s} \in W_i \tag{1}$$

returns the index of the sub-sequence which holds the sample $z_{t_s}$ with the sample time $t_s$.

In each sequence $W$ the minimal $z_{min}$ and the maximal $z_{max}$ sample value is identified. The sample value interval $Z = \langle s_{min}, s_{max} \rangle$ is divided into $k$ disjunct sub-intervals of equal length beginning at $z_{min}$, $Z = Z_0 \cup Z_1 \cup \ldots \cup Z_{k-1}$. This is illustrated in the figure 2.

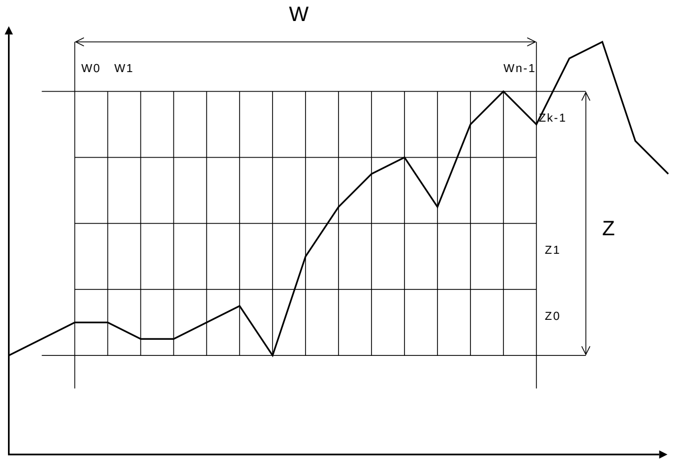

Fig. 2. The sequence of values $W$ is divided into smaller sequences. The value interval $Z$ is divided into smaller intervals. This creates a mesh which is then used for the computation of referential data.

Let the function

$$IdxK(z) : j, z \in Z_j \tag{2}$$

return the index if the sub-interval $Z_j$ where the value of the sample $z$ lies in.

The result of function

$$IdxK_{last}(W_i) = j, z_{last} \in Z_j \tag{3}$$

is the index of one of the sub-intervals of $Z$, $z_{last}$ is the last element of the sequence $W_i$. The function

$$K(W_i, l) = j \tag{4}$$

returns $j = K_{last}(W_i)$ if $\forall z \in W_i \cup W_{i+1} \cup \ldots \cup W_{i+l}$ the prescription $z \in S_{K_{last}(W_i)}$ is true. Otherwise the return value is defined as $j = IdxK(z_e)$ where $z_e \in Z$ is the first value where $z_e \notin S_{K_{last}(W_i)}$.

The function

$$ref(z,l) = \begin{cases} 1 & : \quad IdxK_{last}(W_i) < IdxK(W_i, l) \\ -1 & : \quad IdxK_{last}(W_i) > K(W_i, l) \\ 0 & : \quad IdxK_{last}(W_i) = K(W_i, l) \end{cases} \tag{5}$$

where $z \in W_i$ is used to compute referential data identifying the turning points. It returns the value 1 if the values are going to rise during a certain future period or returns -1 if the prices are going to fall.

The parameters $k$ and $n$ define the delicacy of the mesh. The parameter $k$ impacts the sensitivity of the detected value changes whereas the parameter $n$ impacts the timescale of the function. The parameter $j$ in $ref(x,j)$ defines the horizon of the function. The length of the sequences $W_i$ also impacts the robustness of the predictor. In real world delays often occur which a real system has to deal with. The broker has a certain delay between receiving requests and executing them. The length of $W_i$ impacts the time delay a predictor can be trained on.

## 3.2. Predictor Design

As mentioned before, the training data are not filtered. Additional data are added to the incoming data in order to reduce the computational effort of the training process. The data used to train a predictor have a similar structure as shown in the table 1.

TABLE 1. Training data structure. In this case the predictor can utilize the input values and additional moving averages computed from past 10, 100 and 1000 samples. The underlined values are available to a predictor to determine the turning points at time $t$.

| time | input/bids | ref. | mavg(10) | mavg(100) | mavg(1000) |
|---|---|---|---|---|---|
| $t-8$ | 1.1933 | 0 | 1.193330 | 1.193207 | 1.192460 |
| $t-7$ | 1.1932 | 0 | 1.193310 | 1.193206 | 1.192459 |
| $t-6$ | 1.1933 | 0 | 1.193310 | 1.193207 | 1.192458 |
| $t-5$ | 1.1932 | 0 | 1.193310 | 1.193206 | 1.192457 |
| $t-4$ | 1.1933 | 0 | 1.193310 | 1.193205 | 1.192456 |
| $t-3$ | 1.1934 | 0 | 1.193310 | 1.193206 | 1.192455 |
| $t-2$ | 1.1935 | 0 | 1.193330 | 1.193209 | 1.192454 |
| $t-1$ | 1.1934 | 0 | 1.193330 | 1.193210 | 1.192453 |
| $t$ | 1.1935 | 0 | 1.193350 | 1.193213 | 1.192453 |

At the time $t$ the evolved predictor has to determine whether the input data are at a turning point. It is simple to identify a turning point at the time $t - 3000$ because the samples from time $t - 2999$ to $t$ are present and can be used by the function 5. However we do not have the samples $t+1$ to $t+3000$ to compute the turning point at time $t$ simply because the future has not happened yet.

The task of the predictor is to compute, at the time $t$, the value $p_t$ which can identify a turning point. The values $p_t$ should be in the set $\{-1, 0, 1\}$. The predictor can use a vector of past $m$ samples of the time series $\bar{z}_m(t) = (z_{t-m+1}, \ldots, z_t)$. A vector of the size $h$ of additional supportive values which are computed at the time $t$ is also available $\bar{a}_h(t) = (a_{1t}, \ldots, a_{ht})$. To construct the predictor is to find a function $pred(\bar{z}_m(t), \bar{a}_h(t))$ which is at every time $t$ equal to the hypothetical referential function results; $\bar{z}_m(t)$ is a vector of last $m$ samples measured till the time $t$, $\bar{a}_h(t))$ is the vector of additional data computed for the last sample at time $t$.

Genetic programming [9], [10] (GP) is used to represent a predictor. GP is a methodology to generate computer programs by using evolutionary algorithms (EAs). GP chromosomes are very powerful and can represent any Turing-computable function [14]. The chromosome represents a syntactical tree structure. The leaves (terminals) represent inputs, the nodes (non-terminals) represent functions. The tree is evaluated bottom-up (from the leaves towards the root). The function in the root returns the the total outcome.

To represent the function $pred(\bar{z}_m(t), \bar{a}_h(t))$ by the means of GP a set of terminals representing the function inputs has to exist. A set $\Gamma_{term} = \{f_{z1}, \ldots, f_{zm}\} \cup \{f_{a1}, \ldots, f_{ah}\}$, containing functions returning values from the vectors $\bar{z}_m(t)$ and $\bar{a}_h(t)$, is necessary. It is also useful to add a set of constants $\Gamma_{const} = \{c_1, \ldots, c_n\}$ which can be used for terminal inputs. The set $\Gamma_{func}$ holds the functions assigned to the nodes.

## 3.3. Fitness Function

The fitness value is a measure comparing how good a predictor is when comparing it to the referential function outcomes. The comparison is performed on training data where both input and referential values are available. Suppose there is a sequence $Z = z_{i-n+1}, \ldots, z_i$ of sampled input data. The sequence is translated onto the sequence of referential data $R = r_{i-n+1}, \ldots, r_i$ in the way $r_l = ref(z_l, j), j \in \{i-n+1, \ldots, i\}$. Also the vectors of additional precomputed values are supplied $\bar{A} = \{\bar{a}_{i-n+1}, \ldots, \bar{a}_i\}$. The predictor $pred(\bar{z}_m(t), \bar{a}_h(t))$ is evaluated by creating a sequence of predicted values $P = p_{i-n+1}, \ldots, p_i$, $p_l = pred(\bar{z}_m(l), \bar{a}_l)$.

The following functions can be used for fitness evaluation:

$$E_{mae} = \frac{1}{n+1} \sum_{i=0}^{n} |r_i - p_i| \tag{6}$$

$$E_{mse} = \frac{1}{n+1} \sum_{i=0}^{n} (r_i - p_i)^2 \tag{7}$$

The function above compute the mean average error (6) and the mean square error (7).

TABLE 2. Functions used to compute supportive values in the preprocessor.

| function | description |
|---|---|
| $mavg_{10}$ | moving average over last 10 values |
| $mavg_{100}$ | moving average over last 100 values |
| $mavg_{1000}$ | moving average over last 1000 values |
| $mavg_{10000}$ | moving average over last 10000 values |

TABLE 3. The set of function used in the genome nodes.

| function | result |
|---|---|
| $add(a,b)$ | $a + b$ |
| $sub(a,b)$ | $a - b$ |
| $mult(a,b)$ | $a \times b$ |
| $div_s(a,b)$ | $\begin{cases} a/b & : b \neq 0 \\ a & : b = 0 \end{cases}$ |
| $min(a,b)$ | minimum from $a$ and $b$ |
| $max(a,b)$ | maximum from $a$ and $b$ |
| $avg(a,b)$ | $\frac{a+b}{2}$ |
| $abs(a)$ | $\mid a \mid$ |
| $g(a,b,c)$ | $\begin{cases} c & : a > b \\ 0 & : a \leq b \end{cases}$ |
| $l(a,b,c)$ | $\begin{cases} c & : a < b \\ 0 & : a \geq b \end{cases}$ |

## 3.4. Evolutionary Algorithm

At the beginning of the design process a canonical tree-based GP algorithm was used. The main disadvantage was the tendency to converge all candidate solutions around a specific solution and thus to reduce the adaptability. Therefore the algorithm was replaced with a more sophisticated algorithm – the age-layered population structure (ALPS) algorithm [8] designed by G. S. Hornby. The algorithm divides the population according to the age of their members. It also periodically introduces random solutions to reduce premature convergence. In our case the algorithm was used to work in a dynamically changing environment.

## 4. Experiments

The whole system is implemented in software using the Python and C programming languages. The preprocessor is written in Python because it is much easier to modify the source code. The rest of the system is written in C because it produces more efficient code.

The preprocessor computes the functions in the table 2, the referential function values and adds the results to the data history buffer.

At a single time the evolved GP genomes are free to use last 200 sampled values and all the supportive values attached to the last sample. The genome uses the set of constants $\Gamma_{const} = \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 0, 1, 2, 5, 10, 20, 50\}$. The set of functions $\Gamma_{func}$ is described in the table 3.

To reduce bloat [13] the tree depth is restricted to 5. The crossover and mutation operators are adjusted to deal with the restriction. The population holds 200 members.

The predictor unit communicates with the EA unit. Every generation the EA unit sends the best evolved predictor to the

TABLE 4. The range of prediction accuracies depending on various system settings.

| | |
|---|---|
| predicted correctly | $12 - 31\%$ |
| false predicted rises or falls | $15 - 35\%$ |
| false predicted stagnation | $45 - 70\%$ |

predictor unit where is can replace the worst predictor. The discarded predictor can be send back to the EA unit where it may be used again. The predictor unit can use a variant of a majority function to decide the final outcome of the system.

### 4.1. Results

The system was simulated on sampled EUR/USD data from the year 2005. At the start-up of the system it is better to let the system adapt the predictors before the environment changes for the first time. Then it is able to adapt the predictors on the fly.

The system is written with the support of threads – it can use all cores of a multi-core system. However the system is still painfully slow when it is simulated on a single processor system. Most of the time the system waits for the fitness function evaluation. Therefore the data are dispatched to the system in a lower rate than they would enter the system in a real world application.

The systems performance at the current stage of development is summarized in the table 4. The main problem to deal with is the relative high number of false turning point predictions. There should be as little as possible.

## 5. Conclusions

The results show that the system design is capable, under certain circumstances, of a relatively good prediction rate. The major drawback is the high ratio of mispredicted turning points. Fortunately there is still a lot of work to be done. This work may include tuning the function set or modifying the preprocessor to take advantage of the time which the tick data arrived at.

The acceleration of the fitness evaluation represent another demand to be dealt with. There may be a benefit in the acceleration on modern GPUs [7] or just by using the more conventional approach of parallelization on a cluster machine.

## Acknowledgements

This work was supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research intention No. MSM0021630528 – Security-Oriented Research in Information Technology.

## References

[1] Antonia Azzini, Célia da Costa Pereira, and Andrea Giovanni Battista Tettamanzi. Predicting turning points in financial markets with fuzzy-evolutionary and neuro-evolutionary modeling. In *Applications of Evolutionary Computing*, pages 213–222. Springer, 2009.

[2] George Edward Pelham Box, Gwilym Meirion Jenkins, and Gregory Charles Reinsel. *Time Series Analysis, Forecasting and Control*. Prentice Hall, New Jersey, 2006.

[3] Anthony Brabazon and Michael O'Neill. *Biologically Inspired Algorithms for Financial Modelling*. Springer-Verlag, Berlin Heidelberg, 2006.

[4] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag, New York, 2002.

[5] Jing Dang, Anthony Brabazon, David Edelman, and Michael O'Neill. An introduction to natural computing in finance. In *Applications of Evolutionary Computing*, pages 182–192. Springer, 2009.

[6] Christian Dunis. *Forecasting Financial Markets, Exchange Rates, Interest Rates and Asset Management*. John Wiley and Sons, 1997.

[7] Simon Harding and Wolfgang Banzhaf. Fast genetic programming on gpus. In *EuroGP*, pages 90–101, 2007.

[8] Gregory Scott Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822, New York, NY, USA, 2006. ACM.

[9] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[10] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, Berlin Heidelberg, 2002.

[11] Julian Francis Miller and Peter Thomson. Cartesian genetic programming. In *Proceedings of the 3rd European Conference on Genetic Programing*, Lecture Notes in Computer Science, pages 121–132, Berlin, 1999. Springer Verlag.

[12] Karel Slaný. Branch predictor on-line evolution. In *2008 Genetic and Evolutionary Computational Conference GECCO*, pages 1643–1648. Association for Computing Machinery, 2008.

[13] Terence Soule. *Code Growth in Genetic Programming*. PhD thesis, University of Idaho, 1998.

[14] Astro Teller. Turing completeness in the language of genetic programming with indexed memory. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 136–141, Orlando, Florida, USA, 27-29 1994. IEEE Press.